

# Project - Submission 1 (Group 401)

## 4.1.1 Domain Definition

A PMS (Project Management System) serves as the “spinal-cord” of every company. It helps in managing multiple projects, departments, employees, business partners, etc. In that way, it allows the company to operate efficiently, controlling every area, generating reports, saving contacts and managing all the projects in an organized manner. A good PMS always tries to replicate the stages of the clients operations, be it with different project steps, different interconnected sales and acquisition of material, how the clients relate to the company employees and how those employees are part of different departments.

The most important entity in a PMS is the Company that produces and sells the products, it being the software owner that will provide all the input to the PMS. The Company, as the center point of the software (and the operation) is indirectly connected to all entities present on the software, as it is also, in the real world, connected to all operations it executes. The system enables the company to organize all its workforce on Departments and areas, creating a solid organizational structure that will later be important not just for the daily management of the company but also for strategic decisions such as the allocation of resources and personnel for each Project or even the definition of internal workflows that will define how the company operates on “extra-software” level.

Each department in a company has their own tasks and objectives. A PMS is the key to manage all departments and its employees and mainly, differentiate the sales from those many departments for later analysis, always keeping track of the budget given to each department and their adherence to their proposed strategies. Keep in mind that some Departments might not be related directly to sales, but still, it's important to keep track of their performance through many different metrics (later defined using the data available on the PMS).

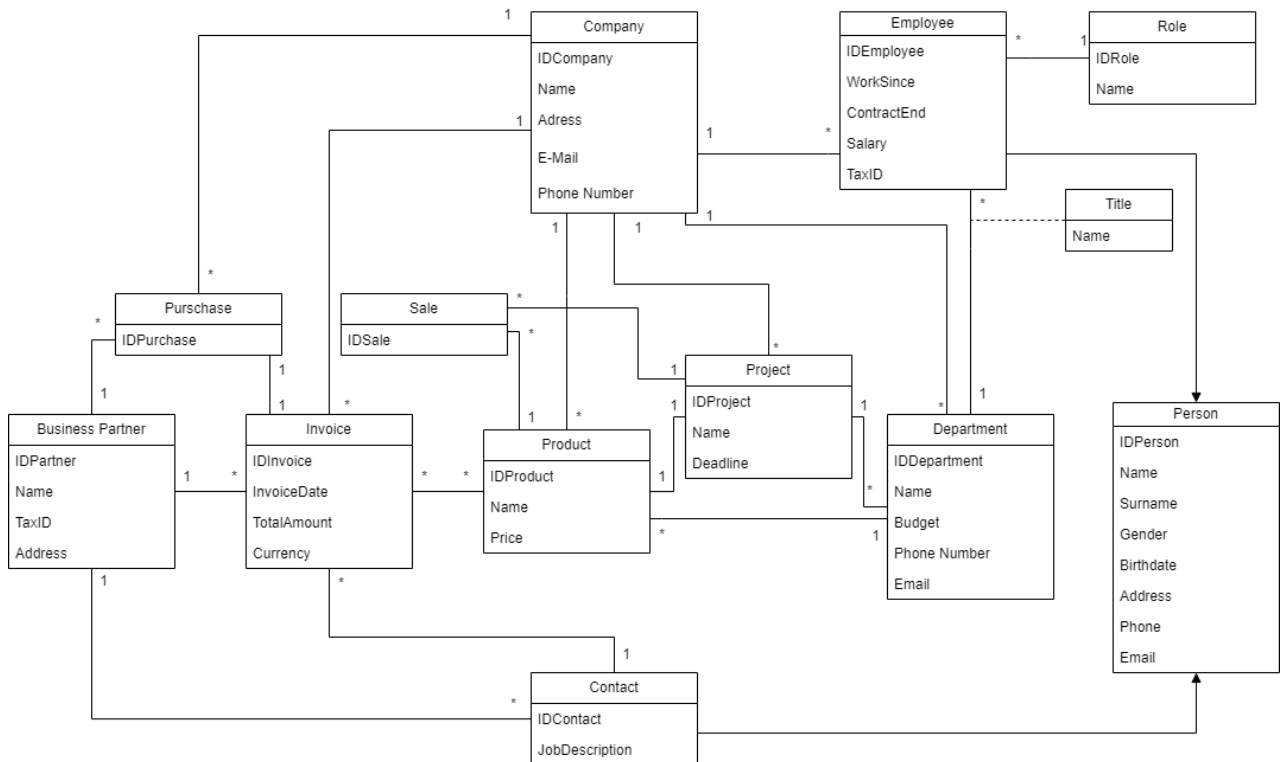
A business is nothing without Sales, therefore a PMS also does not function without them. Sales are the cornerstone of every company and a PMS will help, as it helps with each department, to organize every sale, so the company doesn't lose money due to inefficiency and lack of internal planning. With each sale, a PMS will store the related documents, assign each sale to an Employee/Department and keep track of the profits (or losses) from each sale, creating a whole history of this sale and how it went, creating a comprehensible timeline that can be used for later analysis.

The employees are also an important entity in a PMS as without them the whole operation would freeze. A PMS is important to ensure how each employee is involved in the operation, in which departments they're working and tracking the success of projects they're involved in.

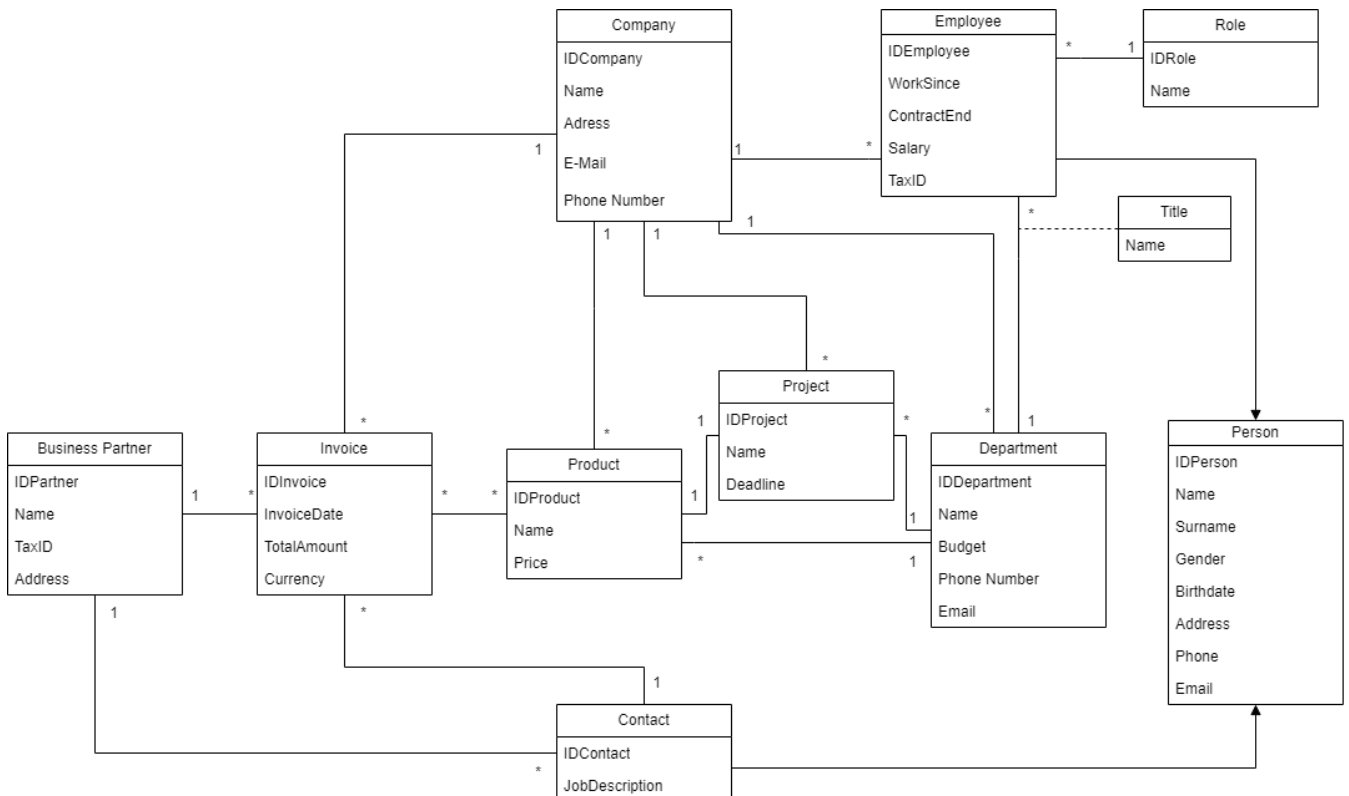
Ultimately, as the name suggests a Project Management System is helpful in turning the management of companies on a seamless task, by connecting the company, its departments, employees, sales, financial documents, customers and general business partners on a robust information web.

## 4.1.2 Initial Conceptual Modeling

### 1. First Conceptual Model



### 2. Second Conceptual Model, after teacher's opinion



Together with the help of a teacher, we came to the conclusion that sale and purchase classes might be redundant, making the relations more complex than they should be, as the prices, quantities and other information related to the sales and purchases of products were already stored on other tables and wouldn't benefit from extra classes. This was our final iteration without the assistance of Generative AI.

### 4.1.3 Generative AI Integration

This interaction was made with ChatGPT 4.o

The idea behind the initial prompt was to generate something completely different from our original approach, so we could decide which to introduce in our Model later on.

#### 1. Initial Prompt

A PMS (Project Management System) serves as the “spinal-cord” of every company. It helps in managing multiple projects, departments, employees, business partners, etc. In that way, it allows the company to operate efficiently, controlling every area, generating reports, saving contacts and managing all the projects in an organized manner. A good PMS always tries to replicate the stages of the clients operations, be it with different project steps, different interconnected sales and acquisition of material, how the clients relate to the company employees and how those employees are part of different departments. The most important entity in a PMS is the Company that produces and sells the products, it being the software owner that will provide all the input to the PMS. The Company, as the center point of the software (and the operation) is indirectly connected to all entities present on the software, as it is also, in the real world, connected to all operations it executes. The system enables the company to organize all its workforce on Departments and areas, creating a solid organizational structure that will later be important not just for the daily management of the company but also for strategic decisions such as the allocation of resources and personnel for each Project or even the definition of internal workflows that will define how the company operates on “extra-software” level. Each department in a company has their own tasks and objectives. A PMS is the key to manage all departments and its employees and mainly, differentiate the sales from those many departments for later analysis, always keeping track of the budget given to each department and their adherence to their proposed strategies. Keep in mind that some Departments might not be related directly to sales, but still, it's important to keep track of their performance through many different metrics (later defined using the data available on the PMS). A business is nothing without Sales, therefore a PMS also does not function without them. Sales are the cornerstone of every company and a PMS will help, as it helps with each department, to organize every sale, so the company doesn't lose

money due to inefficiency and lack of internal planning. With each sale, a PMS will store the related documents, assign each sale to an Employee/Department and keep track of the profits (or losses) from each sale, creating a whole history of this sale and how it went, creating a comprehensible timeline that can be used for later analysis. The employees are also an important entity in a PMS as without them the whole operation would freeze. A PMS is important to ensure how each employee is involved in the operation, in which departments they're working and tracking the success of projects they're involved in. Ultimately, as the name suggests a Project Management System is helpful in turning the management of companies on a seamless task, by connecting the company, its departments, employees, sales, financial documents, customers and general business partners on a robust information web. After this explanation about the software being developed, as a database professional: Create me a UML about what was explained above (remember to maintain a Project class), feel free to add extra classes you may find interesting and explain why

## 2. First Response

To create a UML (Unified Modeling Language) class diagram for the Project Management System (PMS) described, I'll outline the main entities and relationships first. From your explanation, the key entities seem to be:

1. **Company** - The central entity that interacts with all others.
2. **Department** - A subdivision of the company that performs specific tasks and manages resources.
3. **Employee** - Individuals working in departments who contribute to projects and sales.
4. **Project** - A core part of the system, representing the activities the company is working on.
5. **Sales** - Business transactions that generate revenue, assigned to departments and employees.
6. **Customer** - External entities that engage with the company through purchases or contracts.
7. **Document** - Financial or operational records related to projects or sales.
8. **Business Partner** - Other external entities such as suppliers or contractors involved with the company.
9. **Metrics** - Data that tracks performance, which may be tied to departments, employees, projects, or sales.

### Classes and Relationships

#### 1. Company

- Attributes: `company_id`, `name`, `address`, `industry`

- Relationships:
  - Company has many Departments
  - Company has many Projects
  - Company has many Sales
  - Company has many Business Partners

## 2. Department

- Attributes: department\_id, name, budget
- Relationships:
  - Department belongs to Company
  - Department has many Employees
  - Department has many Sales
  - Department has many Projects

## 3. Employee

- Attributes: employee\_id, name, position, salary, hire\_date
- Relationships:
  - Employee belongs to Department
  - Employee works on many Projects (many-to-many with Project)
  - Employee is assigned to many Sales (many-to-many with Sales)

## 4. Project

- Attributes: project\_id, name, start\_date, end\_date, budget
- Relationships:
  - Project belongs to Company
  - Project involves many Employees (many-to-many with Employee)
  - Project generates many Documents
  - Project may involve Business Partners

## 5. Sales

- Attributes: sale\_id, date, amount, profit
- Relationships:
  - Sales belongs to Company
  - Sales involves one or more Departments
  - Sales involves one or more Employees
  - Sales are associated with Customers
  - Sales generates Documents

## 6. Customer

- Attributes: customer\_id, name, contact\_info
- Relationships:

- **Customer** is associated with many **Sales**

## 7. Document

- Attributes: **document\_id**, **type**, **file\_path**, **created\_date**
- Relationships:
  - **Document** is generated by a **Project**
  - **Document** is related to a **Sale**

## 8. Business Partner

- Attributes: **partner\_id**, **name**, **contact\_info**
- Relationships:
  - **Business Partner** works on many **Projects** with **Company**

## 9. Metrics

- Attributes: **metric\_id**, **description**, **value**, **date\_recorded**
- Relationships:
  - **Metrics** are tracked for **Departments**, **Projects**, and **Sales**

## Explanation of Additions

- **Customer**: Essential for tracking external interactions and sales performance.
- **Document**: Added to store important records for both sales and projects, providing traceability.
- **Business Partner**: Companies often work with external suppliers or service providers, so this is important for projects.
- **Metrics**: Metrics allow companies to analyze performance across departments and projects.

I'll now generate a UML diagram based on these ideas.

It seems that the UML generation library I tried to use is not available. I can describe the diagram in more detail, or you could use a tool like Lucidchart, Visual Paradigm, or draw.io to visualize it based on the structure I provided.

Would you like to proceed with a text-based breakdown of how the entities and relationships look in the UML diagram, or should I help guide you through creating it manually with a tool?

## 3. Second Prompt

Can you explain to me why there is a differentiation between Business Partner and a Customer if they can be conceptually overlapped on their operations?

(Here we decide to make questions about the differences spotted between our solution and the AI, it was important to understand the real need for a new class representing a Customer)

## 4. Second Response

The differentiation between a **Business Partner** and a **Customer** in a Project Management System (PMS) is made for clarity and flexibility, despite some overlap in their operations. Here's why separating these entities can be useful:

### 1. Role Differences:

- **Customer:** Typically refers to an entity that **purchases goods or services** from the company. The primary role of a customer is to generate revenue by buying the company's products or services.
- **Business Partner:** Refers to an entity that **collaborates or works alongside the company** to achieve business goals. A business partner can be a supplier, contractor, or another organization involved in the company's operations without necessarily purchasing from it.

Even though in some contexts, a customer might also collaborate with the company in more complex transactions, the two roles have distinct relationships with the company. For example, a **supplier** or **vendor** is a business partner, not a customer, as they provide the materials or services the company needs to produce its products.

### 2. Operation Focus:

- **Customer Operations:** Customers are usually at the **receiving end** of the company's sales processes. The PMS tracks **sales-related metrics** (revenue, profit, satisfaction, etc.) related to customer transactions.
- **Business Partner Operations:** Business partners might engage in **joint ventures, supply chain activities, or long-term collaborations**. The PMS tracks how they contribute to project delivery, resource acquisition, and cost management.

A PMS needs to manage **different types of interactions** with these entities. For example:

- With **Customers**, you focus on handling sales, after-sales service, and customer relationships.
- With **Business Partners**, the focus might be on contract management, joint project execution, or procurement.

### 3. Type of Relationship:

- **Customer:** The relationship is more transactional and usually revolves around specific purchases or services provided. The main goal is to sell products or services.
- **Business Partner:** The relationship is **strategic**, often more collaborative. A business partner could be a long-term supplier, a co-developer of a product, or even a partner in distributing the company's products. This relationship might involve contracts, procurement, and resource-sharing agreements.

#### 4. Flexibility in Reporting:

- By separating the two entities, the PMS allows for **customized reporting and metrics** for each. For instance, customer reports might focus on **sales performance**, **customer satisfaction**, and **profitability**. In contrast, business partner reports would look at **supplier performance**, **resource availability**, and **collaboration outcomes** (e.g., project completion, supply chain efficiency).
- This flexibility helps in **better strategic decision-making** since the performance and success of customers and partners impact the company differently.

#### 5. Potential Overlap:

- **Yes, a customer can sometimes also be a business partner.** For example, a distributor that buys products from the company might also partner to help with marketing or sales expansion. However, keeping them separate ensures that the **different types of interactions** and **roles** they play can be clearly understood and managed.

In essence, while a **Customer** and **Business Partner** might overlap in some scenarios, they generally serve different purposes and require separate management and metrics. The PMS must accommodate both types of relationships to enable comprehensive and effective company operations.

### 5. Third Prompt

Is it important to maintain a Sales table when Invoice Documents already possess the same information available on this table?

(After understanding the differences between a Customer and a Business Partner, again, as seen on the Second Prompt, we ask about the Sales table, which we deleted on our final iteration on the Initial Concept)

### 6. Third Response

Maintaining a separate **Sales** table, even if an **Invoice Document** contains much of the same information, can be important for several reasons. While there is some overlap



between the two, they serve **different purposes** and provide value in distinct ways within a Project Management System (PMS). Here's why it's beneficial to keep them separate:

### 1. Conceptual Difference:

- **Sales:** Represents the **business transaction** or deal itself, capturing high-level details like the product or service sold, the parties involved (customer, employee, department), and the financial outcomes (revenue, profit).
- **Invoice Document:** Represents the **official record** or legal document of the transaction, which is necessary for accounting and compliance purposes. It includes details like taxes, payment terms, and specific line items.

Sales data is usually more focused on **business performance**, while invoice data is more concerned with **financial reporting and legal compliance**. A sales table tracks sales activities, which is broader than just the invoice.

### 2. Analytical Flexibility:

- **Sales Table:** Having a dedicated sales table allows for easier **analysis and reporting** on sales performance, trends, and key metrics such as:
  - Total sales by product or service
  - Sales volume by employee, department, or region
  - Profitability analysis per sale
  - Sales conversion rates (e.g., from opportunities to closed deals)
- These analyses don't necessarily require all the granular financial details an invoice contains, but instead focus on high-level business data. The sales table can be optimized for fast querying and generating reports specific to sales.
- **Invoice Table:** Invoices are more suited for detailed **financial audits** and **billing** purposes. They may contain breakdowns of costs, taxes, discounts, and specific payment terms. If the company needs to assess compliance with tax regulations or track overdue payments, the invoice document is the go-to source.

### 3. Different Lifecycles:

- **Sales Lifecycle:** A sale may exist even before an invoice is generated. For example, a sale might be recorded when the deal is **closed** (contract signed or order placed), but an invoice might only be issued after the product is delivered or the service is rendered.
- **Invoice Lifecycle:** Invoices typically come **later in the process**, after the delivery of goods or completion of services, and are focused on **payment collection**.

If only the invoice table is used, it becomes harder to track the sales **pipeline** or forecast future revenue based on closed deals not yet invoiced.

### 4. Efficiency and Separation of Concerns:

- A **Sales table** allows the system to store simplified, high-level information relevant to the sales team, project managers, or decision-makers without cluttering the table with detailed financial data (e.g., taxes, individual line items) from invoices.
- Meanwhile, the **Invoice Document** table can be maintained by accounting and legal teams, focused on **financial accuracy** and **regulatory compliance**.

#### 5. Handling Complex Transactions:

In some cases, a single **sale** might generate **multiple invoices** (e.g., when a large order is delivered in parts), or a single invoice might be linked to **multiple sales** (e.g., bundled services). A dedicated sales table can help handle these complexities more easily by maintaining a clear record of each sale independent of the invoicing process.

#### 6. Separation of Roles and Responsibilities:

- **Sales Data:** Typically managed by the **sales team** and used for **business insights** and **performance tracking**.
- **Invoice Data:** Managed by the **accounting department**, focusing on **bookkeeping, audits, and legal documentation**.

These two departments have different priorities and workflows. Having separate tables ensures that each department can manage the data that is most relevant to their tasks without interference or confusion.

#### 7. Historical Data and Auditing:

- The **Sales table** can track sales activity even if an invoice has not been issued yet or if a sale is **cancelled, postponed, or modified**. This would allow for more flexible tracking of business development.
- The **Invoice Document** is an immutable legal record. Any updates or cancellations in sales transactions are reflected through **credit notes** or revised invoices, rather than directly altering the invoice.

#### Conclusion:

While an **Invoice Document** contains much of the same information as a **Sales table**, it's still valuable to maintain both. The **Sales table** focuses on business performance, sales trends, and customer relations, while the **Invoice table** focuses on financial accuracy, compliance, and legal requirements. This separation provides greater flexibility, clarity, and efficiency in managing different aspects of the company's operations.

## 7. Fourth Prompt

Introduce a Class Product, make all necessary changes for this inclusion, considering a product can be a product that is sold or a product that is bought by the company for internal use. Also, remove the Metric class, it won't be needed in this context.

(After understanding the importance of the Sales class, we moved on to a few adjustments so the final result would be closer to what we have envisioned. Originally it was supposed to contain a class Product )

## 8. Fourth Response

Let's introduce the **Product** class into the system and make the necessary changes to integrate it effectively. Here's how this affects the overall structure:

### Updated Classes and Relationships

#### 1. Product

- **Attributes:** `product_id`, `name`, `category`, `description`, `price`, `type` (whether the product is "sold" or "bought")
- **Relationships:**
  - A **Product** can be associated with multiple **Sales** (for sold products).
  - A **Product** can be associated with multiple **Projects** (for products bought by the company for internal use, like materials or tools).
  - A **Product** may be procured from a **Business Partner** if bought for internal use.

### Changes to Other Classes

#### 2. Sales

- Now, a **Sale** may involve one or more **Products**, linking products sold to customers.

#### 3. Project

- A **Project** may require one or more **Products** purchased for internal use (e.g., raw materials, machinery, software tools, etc.).

#### 4. Business Partner

- A **Business Partner** may provide products bought by the company.

### Updated UML Overview

- **Product:** A product class is introduced with a distinction between products bought by the company for internal use and products sold to customers.

- **Attributes:** `product_id`, `name`, `category`, `description`, `price`, `type` (bought/sold).
- **Relationships:**
  - Products are sold in **Sales** transactions (many-to-many relationship).
  - Products may be used in **Projects** (many-to-many relationship).
  - Products are procured from **Business Partners** (for those marked as "bought").

Diagram Structure:

Classes:

- **Product** (new)
- **Company**
- **Department**
- **Employee**
- **Project**
- **Sales**
- **Customer**
- **Business Partner**
- **Document** (no change)

Key Relationships:

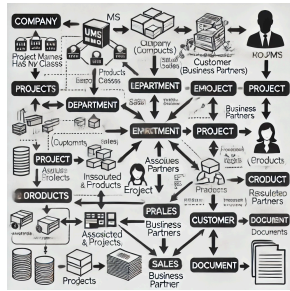
- **Company** "has" many **Products**.
- **Sales** "involves" many **Products**.
- **Project** "requires" many **Products**.
- **Business Partner** "supplies" many **Products** (for internal use).

Would you like me to generate the UML diagram with these updates or walk you through creating it step-by-step?

## 9. Fifth Prompt

Yes

## 10. Fifth Response



Here is the UML diagram that reflects the updated structure of your Project Management System (PMS), including the new **Product** class and its relationships with other entities. Let me know if you'd like any further adjustments!

## 11. Sixth Prompt

Just guide me step by step as in the first answer, do not generate any files or images.

## 12. Sixth Response

Sure! Here's a step-by-step guide to create the updated UML diagram manually, based on the new structure with the **Product** class and removal of the **Metric** class.

## Step 1: Create the Core Classes

## 1. Company

- **Attributes:** `company_id`, `name`, `address`, `industry`
- **Description:** The central entity in the PMS. It is indirectly linked to all other entities.
- **Relationships:**
  - A company **has** multiple **Departments**, **Projects**, **Sales**, **Products**, and **Business Partners**.

## 2. Department

- **Attributes:** `department_id`, `name`, `budget`
- **Description:** Represents the different functional areas of the company.
- **Relationships:**
  - A department **employs** multiple **Employees**.
  - A department **handles** multiple **Sales** and **works on** multiple **Projects**.

### 3. Employee

- **Attributes:** `employee_id`, `name`, `position`, `salary`, `hire_date`
- **Description:** Employees carry out work within departments.
- **Relationships:**
  - An employee **works on** multiple **Projects**.
  - An employee **is assigned to** multiple **Sales**.

#### 4. Project

- **Attributes:** `project_id`, `name`, `start_date`, `end_date`, `budget`
- **Description:** Represents the projects managed within the company.
- **Relationships:**
  - A project **requires** multiple **Products**.
  - A project **involves** multiple **Employees**.
  - A project **generates** multiple **Documents**.
  - A project **involves** multiple **Business Partners**.

### Step 2: Add the Sales and Product Entities

#### 5. Sales

- **Attributes:** `sale_id`, `date`, `amount`, `profit`
- **Description:** Represents sales transactions made by the company.
- **Relationships:**
  - A sale **involves** multiple **Products**.
  - A sale **is associated with** a **Customer**.
  - A sale **is handled by** an **Employee**.
  - A sale **stores** multiple **Documents**.

#### 6. Product

- **Attributes:** `product_id`, `name`, `category`, `description`, `price`, `type` (bought/sold)
- **Description:** Represents products the company either sells or buys for internal use.
- **Relationships:**
  - A product **is sold in** multiple **Sales**.
  - A product **is required by** multiple **Projects**.
  - A product **is supplied by** a **Business Partner** (for products bought internally).

### Step 3: Integrate Customer and Business Partner

#### 7. Customer

- **Attributes:** `customer_id`, `name`, `contact_info`
- **Description:** Represents customers who buy products from the company.
- **Relationships:**
  - A customer **is associated with** multiple **Sales**.

## 8. Business Partner

- **Attributes:** `partner_id`, `name`, `contact_info`
- **Description:** Represents business partners that collaborate with or supply products to the company.
- **Relationships:**
  - A business partner **supplies** multiple **Products** (for internal use).
  - A business partner **is involved in** multiple **Projects**.

## Step 4: Add Document Class

## 9. Document

- **Attributes:** `document_id`, `type`, `file_path`, `created_date`
  - **Description:** Represents the documents generated by the system.
  - **Relationships:**
    - A document **is generated by** multiple **Projects**.
    - A document **is stored in** multiple **Sales**.
- 

## Step 5: Define Relationships

Now that the classes are set up, you can define the key relationships and associations between them:

- **Company:**
  - 1 Company **has** \* Departments, \* Projects, \* Sales, \* Products, 0..\* Business Partners.
- **Department:**
  - 1 Department **employs** \* Employees.
  - 1 Department **handles** \* Sales, **works on** \* Projects.
- **Employee:**
  - 1 Employee **works on** \* Projects.
  - 1 Employee **is assigned to** \* Sales.
- **Project:**
  - 1 Project **requires** \* Products.
  - 1 Project **involves** \* Employees and 0..\* Business Partners.
  - 1 Project **generates** \* Documents.
- **Sales:**
  - 1 Sale **involves** \* Products, **is associated with** 1 Customer, **is handled by** 1 Employee, and **stores** \* Documents.
- **Product:**
  - 1 Product **is sold in** \* Sales.
  - 1 Product **is required by** \* Projects.
  - 1 Product **is supplied by** 1 Business Partner (if bought for internal use).

- **Business Partner:**
    - 1 Business Partner **supplies** \* Products and **is involved in** \* Projects.
- 

### Step 6: Draw the UML Diagram

Once you've defined the classes and their relationships, you can use any UML drawing tool (such as Lucidchart, draw.io, or Visual Paradigm) to visually represent this. Here's a quick outline on how to arrange the elements:

- **Top-Level Classes:** Start with **Company** at the top, with arrows pointing to **Departments**, **Projects**, **Sales**, **Products**, and **Business Partners**.
  - **Department:** Beneath **Company**, showing arrows to **Employees** and **Sales**.
  - **Project:** Show links to **Employees**, **Business Partners**, **Products**, and **Documents**.
  - **Sales:** Show links to **Products**, **Employees**, **Customers**, and **Documents**.
  - **Products:** Connected to both **Sales** and **Projects**, with a link from **Business Partner** (if bought for internal use).
- 

With this, you have a complete conceptual model that you can translate into a visual UML diagram. Let me know if you'd like further clarifications!

## 4.1.4 Final Conceptual Modeling

Our final conclusion, after the use of the AI model was to reimplement the class Sale and to change Project attributes, adding a StartDate and EndDate, while removing Deadline.



